(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 555 997 A2**

# (12) EUROPEAN PATENT APPLICATION

(21) Application number : **93300842.7**

(51) Int. Cl.$^5$ : **H04L 29/06**

(22) Date of filing : **04.02.93**

(30) Priority : **10.02.92 US 830291**

(43) Date of publication of application :
**18.08.93 Bulletin 93/33**

(84) Designated Contracting States :
**DE ES FR GB IT SE**

(71) Applicant : **AMERICAN TELEPHONE AND TELEGRAPH COMPANY**
**32 Avenue of the Americas**
**New York, NY 10013-2412 (US)**

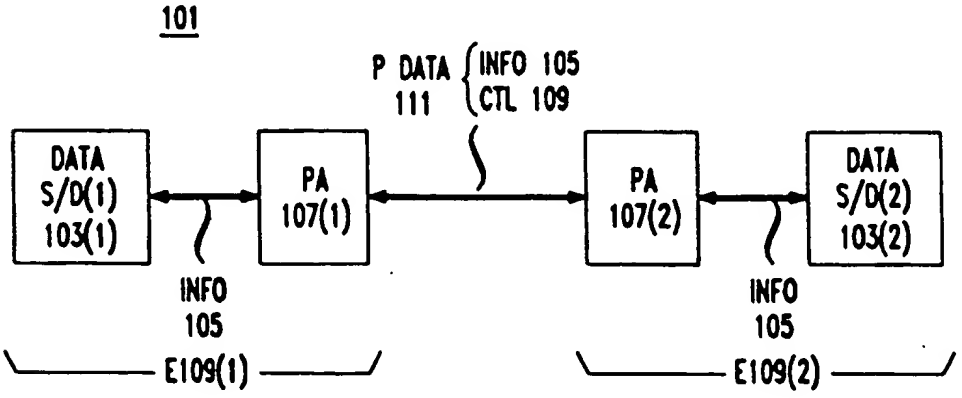
(72) Inventor : **Holzmann, Gerard Johan**
**15 Chestnut Hill Drive**
**Murray Hill, New Jersey 07974 (US)**

(74) Representative : **Watts, Christopher Malcolm Kelway, Dr. et al**
**AT & T (UK) Ltd. 5, Mornington Road**
**Woodford Green Essex, IG8 0TU (GB)**

## (54) Apparatus and methods for implementing protocols.

(57) Apparatus and methods for communicating using protocols. The apparatus and methods employ protocol descriptions written in a device-independent protocol description language. A protocol is executed by employing a protocol description language interpreter to interpret the protocol description. Communication using any protocol for which there is a protocol description may be done by means of a general protocol. The general protocol includes a first general protocol message which includes a protocol description for a specific protocol. The protocol apparatus which receives the first protocol message employs a protocol description language interpreter to interpret the included protocol description and thereby to execute the specific protocol. The protocol apparatus may also be made to adapt to its environment by encaching protocol descriptions which were received in an earlier first general protocol message and interpreting an encached protocol description in response to a second general protocol message which includes a protocol identifier specifying the encached protocol description.

FIG. 1




EP 0 555 997 A2

means capable of interpreting the protocol description language to interpret the protocol description as required to communicate using the specific protocol.

In still another aspect, the invention is protocol apparatus for communicating in distributed system, the apparatus including:

. means for receiving a first general protocol message, the first general protocol message including a protocol description which describes a specific protocol and which employs a protocol description language which is independent of any particular implementation of the protocol apparatus; and

. means for responding to the first general protocol message which are capable of interpreting the protocol description language and which interpret the protocol description as required to communicate using the specific protocol.

In a further aspect, the invention is apparatus for communicating in a distributed system, the apparatus including:

. first protocol apparatus for communicating using a general protocol and

. second protocol apparatus for communicating using the general protocol,

. the first protocol apparatus including

. means for providing a first general protocol message which includes a protocol description which describes a specific protocol and which employs a protocol description language which is independent of any particular implementation of the second protocol apparatus; and

. means for employing the specific protocol to communicate with the second protocol apparatus after providing the first general protocol message; and

. the second protocol apparatus including

. means for receiving the first general protocol message from the first protocol apparatus; and

. means for responding to the first general protocol message which are capable of interpreting the protocol description language and which interpret the protocol description as required to communicate using the specific protocol.

The foregoing and other aspects, objects and advantages of the invention will be apparent to one of ordinary skill in the art who peruses the following Drawing and Detailed Description, wherein:

## Brief Description of the Drawing

FIG. 1 is a block diagram of a typical system in which protocols are used;
FIG. 2 is a block diagram of a first apparatus incorporating the invention;
FIG. 3 is a block diagram of a second apparatus incorporating the invention;
FIG. 4 is a table of instructions in a protocol description language;
FIG. 5 is a flowchart of the bootstrap state;
FIG. 6 is a flowchart of the error state;
FIG. 7 is a state diagram for the alternating bit protocol;
FIG. 8 is a protocol description for the receive side of the alternating bit protocol;
FIG. 9 is a protocol description for the send side of the alternating bit protocol;
FIG. 10 is a fragment of the transition procedure;
FIG. 11 is a protocol description for the bootstrap state;
FIG. 12 is a protocol description for the error state; and
FIG. 13 is a block diagram of an embodiment which encaches downloaded protocol descriptions.

The reference numbers employed in the Drawing and the Detailed Description have three or more digits. The two least significant digits are a number within a figure; the remaining digits are the figure number. Thus, the element with the reference number "305" is first shown in FIG. 3.

## Detailed Description

The following Detailed Description will first provide an overview of the techniques of the invention and will then disclose in detail how the Alternating Bit Protocol (described at Holzmann, *supra*, pp. 75-77) may be implemented using the techniques of the invention.

### Overview: FIGs. 1-3

FIG. 1 is a block diagram of a system 101 in which protocols are used for communication between a first entity 109(1) and a second entity 109(2). Each entity 109 includes a source or destination 103 for information (INFO) 105 and a protocol apparatus 107.

mentations of protocol apparatus 201 will implement different versions of the protocol. Second, protocol description 203 is written only once, but will be used many times. It is therefore worthwhile to expend great efforts to ensure that protocol description 203 is in fact a correct, complete and unambiguous description of the protocol. Third, the part of protocol apparatus 201 which may differ in the different implementations is protocol execution device 207. However, protocol execution device 207 must now only be able to correctly execute the protocol instructions 205 in protocol description 203. That is, the problem is no longer the correct implementation of the protocol, but rather the correct implementation of an instruction set in a single device. This problem is, however, far better understood than the problem of implementing a protocol in two devices, and consequently, implementations of the protocol instruction set in different protocol apparatuses 201 are far more likely to be correct than implementations of the protocol itself.

## Apparatus for Executing a General Protocol: FIGs. 3 and 13

Perhaps the most significant advantage of protocol apparatus 201 is that it can execute *any* protocol for which there is a protocol description 203 written in the protocol description language. Consequently, protocol apparatus 201 can easily be modified to make a general protocol apparatus which executes a *general protocol* and which can therefore dynamically execute any protocol for which there is a protocol description 203. The general protocol is simply the following:

- in a sending protocol apparatus, sending a general protocol message which includes a protocol description 203;
- in a receiving general protocol apparatus, employing protocol instruction interpreter 209 to execute the protocol description 203 contained in the general protocol message.

FIG. 3 shows such a general protocol apparatus 301. General protocol apparatus 301 includes protocol instruction interpreter memory (PIIM) 309, which contains protocol description 203 for the protocol currently being executed by protocol apparatus 301 and protocol instruction interpreter data (PIIDATA) 311, which is data employed by protocol instruction interpreter 209 in executing protocol description 203. Protocol interpreter 209 has two additional components: bootstrap component (BOOT) 305 and error component (ERR) 307. These components make it possible for general protocol apparatus 301 to execute the general protocol, and thereby make it possible for any protocol apparatus 107 which can provide a protocol description 203 to protocol apparatus 301 to use the protocol described in the protocol description 203 to communicate between the entities 109 to which protocol apparatus 107 and protocol apparatus 301 belong. Of course, both protocol apparatuses involved in the communication may be general protocol apparatuses 301.

Protocol apparatus 301 executes the general protocol as follows: bootstrap 305 listens for a general protocol message (indicated by arrow 313) from the other protocol apparatus. In a preferred embodiment, the general protocol message uses the same path between the protocol apparatuses as does protocol data 111. In other embodiments, there may be a special path for the general protocol message. The general protocol message further contains at least the first part of protocol description 203 for the specific protocol to be executed. When bootstrap 305 receives the general protocol message, it loads the message into a buffer in protocol instruction interpreter data 311 and performs checks as described below. If the message passes the checks, bootstrap 305 loads the general protocol message into the portion of memory 309 reserved for protocol description 203. Thereupon, interpreter 209 begins executing the protocol instructions 205 in the message, beginning with the initial instruction. If protocol description 203 is longer than the maximum size of an general protocol message, then the first part of protocol description 203 contains protocol instructions which, when executed, cause the rest of protocol description 203 to be loaded.

In a preferred embodiment, the general protocol requires that the general protocol message contain checking information which permits error checking and protocol data information which indicates how protocol instruction interpreter 209 is to interpret protocol data 111 and that the receiving general protocol apparatus 301 use the checking information and the protocol data information. In the preferred embodiment, there are two items of checking information: a checksum for the general protocol message and a required first instruction. On receiving the general protocol message, bootstrap 305 computes the general protocol message's checksum and compares it with the checksum in the message; if they are different, there has been a transmission error and bootstrap 305 waits for another general protocol message. If bootstrap 305's check of the required first instruction in the general protocol message indicates that the general protocol message is not a protocol description 203, the error component 307 of protocol instruction interpreter 209 returns an error message (indicated by arrow 315) to the protocol apparatus 101 which provided the general protocol message. Thereupon, error 307 waits for a valid general protocol message. Once the general protocol message has been successfully received, it is executed by protocol instruction interpreter 209, and as part of the execution, the protocol data information in the general protocol message is used to set parameter values in protocol instruction interpreter

scription.

As will be explained in more detail below, the protocol descriptions 203 employed in a preferred embodiment define a finite state machine. Consequently, a given protocol description 203 is divided into a set of numbered states (S)1321. To permit location of the states, protocol description 1311 is divided into two parts: protocol description body (PDB) 1314, which contains the instructions for the states, and state table 1313, which relates state numbers to the locations of the corresponding states 1321. There is an entry 1315 in state table 1313 for each state 1321 in the protocol description body, and each entry contains the state number (SN) 1317 and the offset (OFF) 1319 of that state from the beginning of protocol description 1311.

The modifications required in bootstrap 305 will be immediately apparent from FIG. 13 and the description of the general protocol for general protocol apparatus 1323. When a general protocol message is received which contains a protocol description identifier for which the protocol description1311 is in memory 1302, bootstrap 305 simply causes interpreter 209 to begin executing the specified protocol description; otherwise, bootstrap 305 retains the identifier from the general protocol message and causes error 307 to return an error message and wait for a message which contains the protocol description 1311. When the message arrives, error 307 causes bootstrap 305 to compare the retained identifier with the identifer in the general protocol message containing the protocol description 1311, and if they agree, bootstrap 305 places the protocol description 1311 in memory 1302 and makes an entry 1305 for the new protocol description 1311 in protocol description table 1303.

Of course, many variation on the above arrangements are possible. For example, memory 1302 is necessarily finite; consequently, bootstrap 305 may have to remove one protocol description 1311 to make room for another. One way of doing this would be to include size and most recent use information in protocol description table 1303, and bootstrap 305 could use that information to determine which protocol descriptions 1311 should be removed. Further, the general protocol for general protocol apparatus 1323 might include a checksum in the general protocol message for the protocol description 1311 identified by the identifier. Bootstrap 305 could use the checksum to make sure that the copy of the protocol description 1311 in memory 1302 was the same as the copy held by the sender. If it was not, bootstrap 305 could send an error message requesting the protocol description and then proceed as previously described for protocol descriptions for which there was no copy in memory 1302.

## Implementation of Protocol Apparatus 301

A prototype implementation of protocol apparatus 301 has been constructed in which protocol execution device 207 is a computer capable of executing programs written in the well-known "C" programming language. In the prototype implementation, protocol instructions 205 belonging to a protocol description language are interpreted by a protocol instruction interpreter which is written in C and is executed by a process running on the computer. General protocol apparatus 301 has been tested by writing a protocol description 203 for the alternating bit protocol in the protocol description language and executing the protocol by executing the protocol description 203. The following discussion will first disclose the protocol description language, then protocol interpreter 209, bootstrap 305, and error component 307, and finally protocol description 203 for the alternating bit protocol.

## The Protocol Description Language: FIG.4

FIG. 4 shows the instructions in a preferred embodiment of protocol description language 401. The instructions fall into two classes: those which perform general stack management and expression evaluation, shown in table 403, and those which perform operations which are particularly related to the execution of protocols, shown in table 405.

As is apparent from table 403, protocol instruction interpreter 209 is a stack machine. The stack, maintained in protocol instruction interpretation data 311, is a standard integer size push-down stack. The PUSH_BYTE and PUSH_WORD instructions permit data to be pushed onto the push-down stack. The other instructions take their operands and parameters from the top of the stack and push their results back onto the top of the stack. When a stack overflow or underflow occurs, interpreter 209 ceases executing the protocol, error component 307 sends an error message to the other protocol apparatus 107, and error component 307 then waits for an error handling message from the other protocol apparatus 107. Of course, how the other protocol apparatus 107 responds to the error message is part of the protocol described in protocol description 203. The same thing happens if an arithmetic error such as a zero divide or an integer overflow occurs.

The functions of the instructions in table 405 are generally clear from the table; however, certain instructions require a more detailed explanation. Beginning with the instructions in finite state machine control 407,

contains the current byte position in the state being executed. At 1006, prot is set to the value of cur_state, so that execution starts at the first byte of the state. The while loop indicated at 1007 continues to execute as long as prot has a non-0 value, i.e., essentially until a return statement transfers control out of transition.

The body of the while loop is a switch statement which contains a case for each of the instructions in protocol description language 401. On each iteration of the loop, the variable prot is incremented by 1, so that it points to the next byte in the state. The value of that byte determines which case is executed. If there is no case corresponding to that value, default case 1015 is executed, which puts interpreter 209 into the error state and thereby transfers control to error 307. Where required, a case further contains debugging code and assertions to check whether requirements for the execution of the instruction are fulfilled. If interpreter 209 is only used with fully tested and verified protocol descriptions 203, the assertions and debugging code may be disabled.

Fragment 1001 shows two cases in addition to default case 1015: those for NXT and I_NXT. With NXT 1011, the case simply pops the value at the top of the stack (i.e., the next state), checks whether the value is in the range of values for states, and returns the value. With I_NXT, the value of the next state is in the code, and not on the stack, so the case increments prot by one, checks whether the value at that point in the code is within the range, and returns the value.

**Implementation of Bootstrap 305:FIG.5**

In a preferred embodiment, Bootstrap 305 is implemented as a state of interpreter 209. Unlike the other states, which are defined by the protocol description loaded in by bootstrap 305, bootstrap 305 and error 307 are required for the execution of the general protocol and therefore must be built into a protocol apparatus 301 *before* a protocol description 203 can be loaded.

Since these two states are required for the general protocol, they are the only ones that enforce a predefined format on incoming messages, and that must require, the presence of certain kinds of data to permit checking of the general protocol message. As soon as these two states have successfully received a general protocol message with protocol description 203, they hand off control of the general protocol apparatus to the protocol description 203.

In a preferred embodiment, bootstrap 305 is implemented with a single call run (BOOTSTRAP). Procedure run () is the implementation of interpreter 209 in a preferred embodiment. The procedure is reproduced completely below.

```
run(S)
{       int n = s;
        while (n >= 0 && n <= SMAX && State[n].cont)
                n = transition(n);
        return n;
}
```

run is a loop which invokes the procedure transition with a state number. transition then puts interpreter 209 into the proper state of protocol description 203 or the states which implement bootstrap 305 or error 307. The loop ceases execution when a value which is out of range of the legal state numbers is received. Thus, when invoked with BOOTSTRAP, a constant indicating the bootstrap state, the run simply puts interpreter 209 into the bootstrap state.

Most of the concepts involved in implementing an embodiment of protocol apparatus 301 can be illustrated using an implementation of bootstrap 305. In a protocol apparatus 301 employing such an implementation, the code for bootstrap 305 would always be present in protocol instruction interpreter memory 309.

For a general understanding of bootstrap 305, the flow chart of FIG.5 suffices. As shown in oval 503, bootstrap implementation 501 waits for the general protocol message from the remote protocol apparatus 107. When the message comes, it is loaded into a buffer in protocol instruction interpreter data 311. Next, the message is checked. First, a checksum is performed, to make sure the message is uncorrupted. If the checksum is non-zero, a transmission error has occurred, and the machine returns to the start of the bootstrap state (diamond 505, 507). If the checksum is zero, a check is made if the message has the correct type (diamond 509). In a preferred embodiment, the first instruction is required to be the definition of the BYTEORDER for the lower interface. This byte-order definition specifies the order in which the bytes in a word sent according to the protocol are transmitted across the lower level interface: most or least significant byte first. It need not match the byte-order used in either the receiving or the sending entity. If the message is not a valid protocol description 203, interpreter 209 enters error 307 (511).

If the message is a valid protocol description 203, the contents of receive buffer is assigned to the initial state of the newly loaded protocol, and control is transferred to that state (box 515). A full implementation 1101

at byte 18 is reserved for the checksum which bootstrap 305 uses to check for correctness of transmission.

Portion 807 is state 1 of the portion of the finite state machine which receives messages. In this state, the finite state machine waits for a message (byte 0). When the message arrives, it is placed in RBUF. At bytes 2 through 12, the finite state machine writes a value indicating an acknowledgment (in this case, the character 'A') into the transmittal buffer, obtains the sequence number in the last byte of RBUF, copies the sequence number into TBUF following the 'A', and sends the acknowledgment. At bytes 14 through 20, the finite state machine compares the value of the sequence number retained in VAR_E with the sequence number in the last byte of RBUF. If they are the same, indicating that the received message had the right sequence number, bytes 23 through 33 are executed; otherwise, state 1 is again executed from the beginning, as shown at byte 34. In bytes 23 through 31, the sequence number saved in VAR_E is subtracted from 1 and the result saved in VAR_E again. Then, the message is sent to its destination and state 1 is again executed from the beginning.

FIG. 9 shows how the portion of the alternating bit protocol which sends messages is implemented in a preferred embodiment. In the preferred embodiment, protocol apparatus 107 in which send portion 901 is implemented is a protocol apparatus 201 or 301 which is sending to a protocol apparatus 301. Consequently, the send portion is implemented in protocol description language 401 and includes instructions which download receive portion 801 to the receiving protocol apparatus 301.

The buffers and variables used in portion 901 are defined in part 803 of FIG. 8. In the prototype, buffers 0 and 1 are preset to each contain a message; as may be seen from part 809 of FIG. 8, buffer 0 (named M0) is set to the ASCII character 'M' with the sequence number "0" appended to it, while buffer 1 is set to the ASCII character 'M' with the sequence number "1" appended to it. The buffer R_run contains the code of portion 807, while the buffer R_ini contains the code of portion 805. The buffer R_ack, finally, is used for the acknowledgement received from receiver 801. There are two variables: VAR_S, which holds the sequence number which is to be attached to the message, and VAR_CNT, which is a count of the number of bytes sent by the sender.

Returning to FIG. 9, the allocation and initialization of the sender buffers and variables defined in 803 takes place in section 903, which is state 0. In bytes 0-13, VAR_S and VAR_CNT are both allocated and set to 0; in bytes 14 and 15, receiver initialization code 805, contained in the buffer R_ini, is sent to the receiver; in bytes 16 and 17, the code 807 for state 1 of the receiver, contained in the buffer R_run, is sent to the receiver. These lines 905 thus perform the downloading of protocol description 203 to the receiving protocol apparatus 301. At byte 18, finally, the I_NXT instruction starts execution of state 1 907.

At bytes 0-2 of state 1 907, the current value of VAR_S is pushed onto the stack. At byte 3, SEND takes its parameter from the top of the stack; thus, if VAR_S has the value 0, the message in buffer M0 is sent; if it has the value 1, the message in buffer M1 is sent. In an embodiment for use in an actual communications system, there would be code preceding the SEND instruction which employed the OBTAIN instruction to obtain a byte of data to be sent and place the data in buffer M0 or M1, depending on the value of VAR_S, and then employed CPY_BYTE to append "0" or "1" to the data, again depending on the value of VAR_S.

The code in bytes 4-8 receives the acknowledgment from the receiver and pushes it onto the stack. As pointed out in the discussion of the receiver, the acknowledgment has the form 'A'<sequence_number>. The top byte of the stack consequently should contain 'A' and the next byte should contain the sequence number. In bytes 9-11, the top byte is checked to see whether it contains 'A'. If it does, bytes 14-31 are executed; otherwise, execution continues at byte 32. Bytes 14-31 check whether the acknowledgement has the right sequence number; if it does, they set VAR_S to the next sequence number. More specifically, bytes 14-20 check whether the sequence number in the acknowledgment is the same as the value of VAR_S. If it is not, execution continues at byte 32; if it is, VAR_S is set to its new value by subtracting its current value from 1 (bytes 23-31).

The code in bytes 32-54 updates VAR_CNT and terminates state 1 if the number of messages is greater than the constant NR_MSGS, defined in 803 to be 32765. In bytes 32-40, 1 is added to the current value of VAR_CNT. In bytes 41-47, VAR_CNT is pushed onto the stack, the most and least significant bytes of NR_MSGS is pushed onto the stack, and the two values are compared. If VAR_CNT>=NR_MSGS, bytes 50-52 put the value -1 on the stack. NXT returns that value to run, which thereupon terminates, as previously explained. Otherwise, byte 54 is executed, which causes state 1 907 to again begin execution.

## Performance of Protocol Apparatus 201 or 301

The performance of the implementation of the alternating-bit protocol just described was compared with the performance of an implementation in which the sender and receiver were simply coded in the "C" programming language. The extra overhead caused by the use of protocol description 203 and interpreter 209 instead of a "C" program ranged from 10-30%, depending on the length of the message being transmitted (longer messages have the lower overhead). In many applications, the extra overhead will be offset by the fact that the protocol apparatus of the invention can interpret *any* protocol for which there is a protocol description 203. Fur-

apparatus; and

employing the specific protocol to communicate with the first protocol apparatus.

7. The method set forth in claim 6 further characterized by:

in the first entity,

receiving a second general protocol message which includes a protocol specifier specifying the protocol description;

responding to the second general protocol message by determining whether the specified protocol description is accessible to the first entity;

if the specified protocol description is accessible, employing the first protocol description interpretation means to interpret the specified protocol description; but

if the specified protocol description is not accessible, sending a first error message and thereupon performing the step of receiving the first general protocol message.

8. Protocol apparatus (301) for communicating in a distributed system, the apparatus being characterized by:

means (305) for receiving a first general protocol message, the first general protocol message including a protocol description (203) which describes a specific protocol and which employs a protocol description language which is independent of any particular implementation of the protocol apparatus; and

means (209) for responding to the first general protocol message which are capable of interpreting the protocol description language and which interpret the protocol description as required to communicate using the specific protocol.

9. The protocol apparatus set forth in claim 8 further characterized in that:

the means for receiving further checks the first general protocol message.

10. The protocol apparatus set forth in claim 8 further characterized in that:

the first general protocol message includes a parameter (415) for interpreting protocol data transferred according to the specific protocol; and

the means for responding to the first general protocol message interprets the protocol data according to the parameter.

11. The protocol apparatus set forth in claim 8 further characterized in that:

the protocol apparatus further includes error handling means (307);

the means for receiving the first general protocol message further receives a second general protocol message including a protocol specifier (1307) specifying the protocol description; and

the means for responding to the first general protocol message further responds to the second general protocol message by determining whether the specified protocol description is accessible and if the specified protocol description is accessible, interpreting the specified protocol description, but if the specified protocol description is not accessible, sending a first error message.

12. Apparatus for communicating in a distributed system, the apparatus being characterized by:

first protocol apparatus (301) for communicating using a general protocol and

second protocol apparatus (301) for communicating using the general protocol,

the first protocol apparatus including

means (209,903) for providing a first general protocol message which includes a protocol description (203) which describes a specific protocol and which employs a protocol description language which is independent of any particular implementation of the second protocol apparatus; and

means (209,907) for employing the specific protocol to communicate with the second protocol apparatus after providing the first general protocol message; and

the second protocol apparatus including

means (301) for receiving the first general protocol message from the first protocol apparatus; and

means (209) for responding to the first general protocol message which are capable of interpreting the protocol description language and which interpret the protocol description as required to communicate using the specific protocol.

13. Peripheral apparatus for communicating information using a protocol, the apparatus being characterized by:

means (111) for transferring the information according to the protocol;

FIG. 1

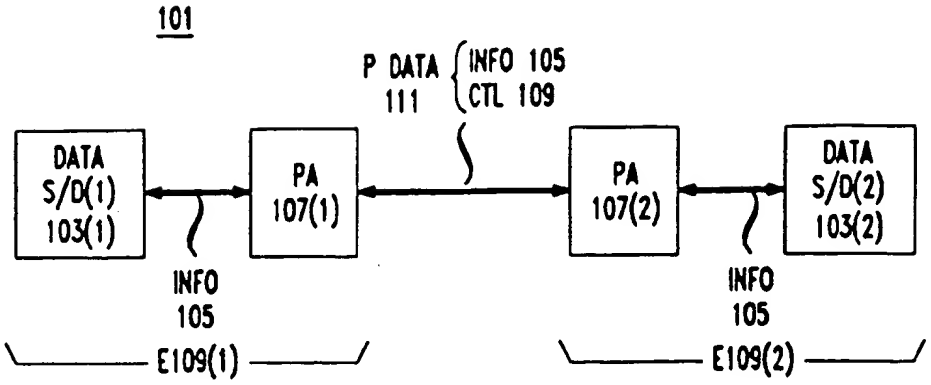



FIG. 2

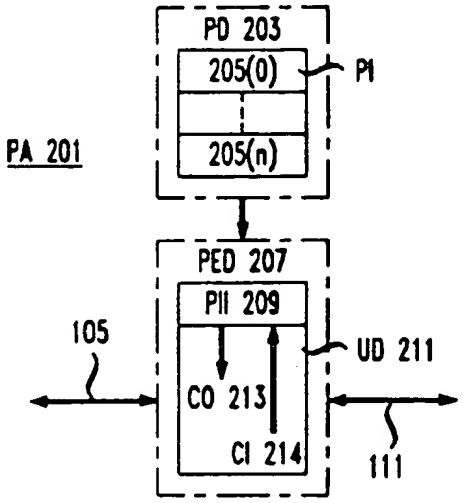
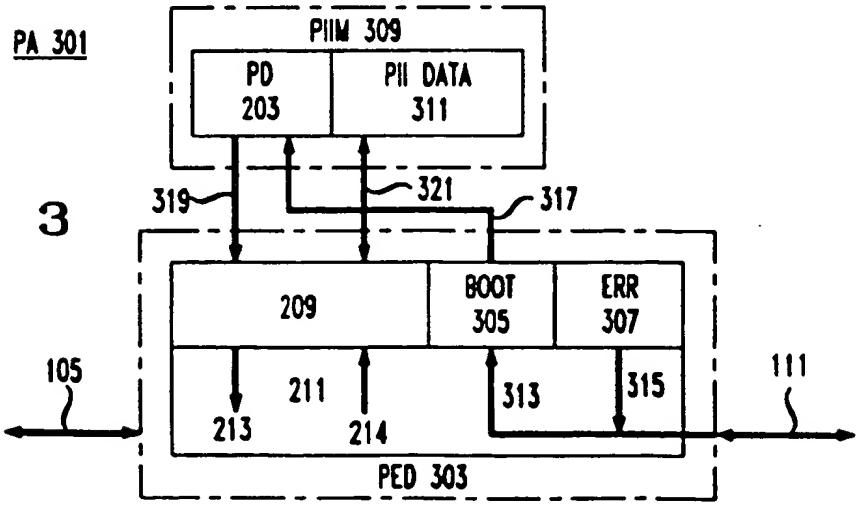



FIG. 3






EDOCID: <EP 0555997A2>

FIG. 5

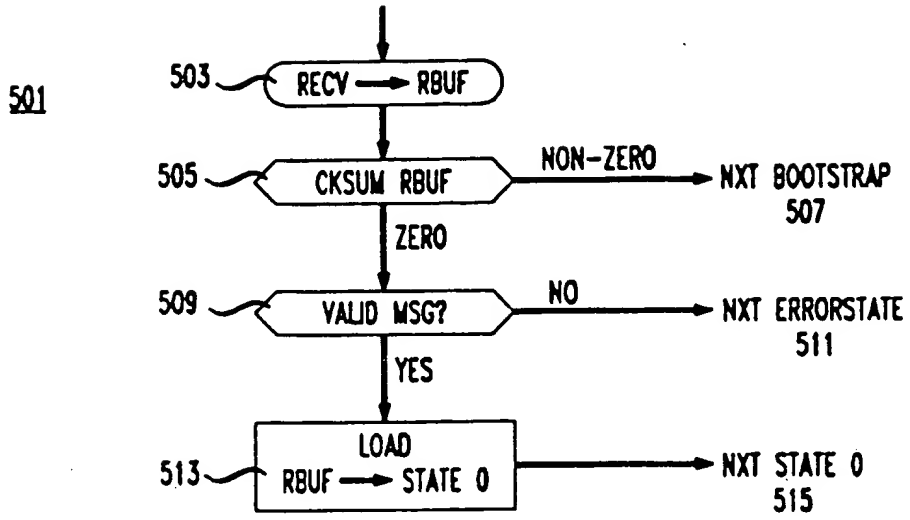
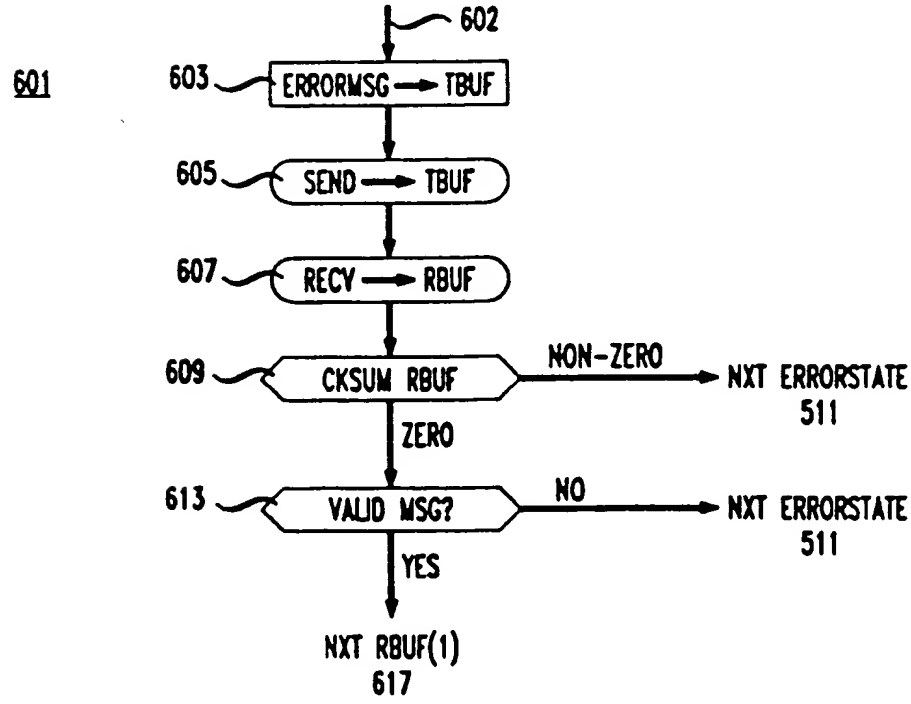



FIG. 6



# FIG. 8

```
/* Receiver Buffers */
#define RBUF     0        /* receive buffer */
#define TBUF     1        /* transmit buffer */
#define VAR_E    2        /* variable 'e' - receiver side */

/* Transmitter Buffers */
#define M0       0        /* message M0 */
#define M1       1        /* message M1 */
#define R_run    2        /* Abp_rcv_run     */
#define R_ini    3        /* Abp_rcv_ini     */
#define R_ack    4        /* receive buffer - for acks     */
#define VAR_S    5        /* variable 's' - sender side    */
#define VAR_CNT  6        /* variable 'cnt' - sender side  */

#define B_O      1        /* byteorder */
#define NR_MSGS 32765     /* number of test messages sent */
```

803

```
BYTE Abp_rcv_ini[] = {          /* initialization Buf[R_ini]; recvd in State[0] */
/*0*/   BYTEORDER,        B_O,
/*2*/   I_ALLOC,          TBUF, 2,
/*5*/   I_SETSIZE,        TBUF, 2,
/*8*/   I_ALLOC,          VAR_E, 1,
/*11*/  I_RECV,           RBUF,
/*13*/  I_LOAD,           1, RBUF,/* input becomes State[1] */
/*16*/  I_NXT,            1,      /* execute it */
/*18*/  0,                0,      /* room for the checksum; required on 1st msg */
};
```

805

```
BYTE Abp_rcv_run[] = {  /* abp receiver Buf[R_run]; recvd in State[1] */
/*0*/   I_RECV,           RBUF,
/*2*/   I_CPY_BYTE,       TBUF,   0,      'A',
/*6*/   I_PUSH_BYTE_VAR,          RBUF,   1,
/*9*/   H_CPY_BYTE,       TBUF, 1,
/*12*/  I_SEND,           TBUF,
/*14*/  I_PUSH_BYTE_VAR,          RBUF,   1,
/*17*/  I_PUSH_BYTE_VAR,          VAR_E,  0,
/*20*/  EQ,
/*21*/  IF,     34,               /* e == rbuf[1] */
/*23*/          I_PUSH_BYTE,      1,
/*25*/          I_PUSH_BYTE_VAR,VAR_E,  0,
/*28*/          MINUS,
/*29*/          H_CPY_BYTE,       VAR_E,  0,
/*32*/          I_ACCEPT,         RBUF,
/*34*/  I_NXT,  1                 /* stay in same state */
};
```

807

```
BYTE Msg0[] = { /* message Buf[M0], received in Buf[RBUF] */
        'M', 0
};
BYTE Msg1[] = { /* message Buf[M1], received in Buf[RBUF] */
        'M', 1
};
```

809

801

# FIG. 10

1003

```
transition(n)
{       BYTE b1, *cur_state = State[n].cont;
        static   int Stack[SMAX+1];
        register BYTE *prot;        1005
        register int w0, w1, w2, i;
        register int *sp = &Stack[SMAX];

        prot = cur_state;                    1006
1007    while (prot) {                       1009
                switch (*prot++) {

                /***** FSM CONTROL *****/

                case NXT:
                        assert(sp < &Stack[SMAX]);
                        w0 = POP;
1011                    debug("next %d0, w0, 0, 0, 0);
                        if (w0 < 0 || w0 > SMAX || !State[w0].cont)
                                return ERRORSTATE;
                        return w0;
                case I_NXT:
                        w0 = *prot++;
1013                    debug("next %d0, w0, 0, 0, 0);
                        if (w0 < 0 || w0 > SMAX || !State[w0].cont)
                                return ERRORSTATE;
                        return w0;
                ...

                default:
1015                    debug("Error <%d>0, *prot, 0, 0, 0);
                        return ERRORSTATE;
        }       }
}
```

1001

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 555 997 A3**

# (12) EUROPEAN PATENT APPLICATION

(21) Application number : **93300842.7**

(51) Int. Cl.[5] : **H04L 29/06**

(22) Date of filing : **04.02.93**

(30) Priority : **10.02.92 US 830291**

(43) Date of publication of application :
**18.08.93 Bulletin 93/33**

(84) Designated Contracting States :
**DE ES FR GB IT SE**

(88) Date of deferred publication of search report :
**15.02.95 Bulletin 95/07**

(71) Applicant : **AT & T Corp.**
**32 Avenue of the Americas**
**New York, NY 10013-2412 (US)**

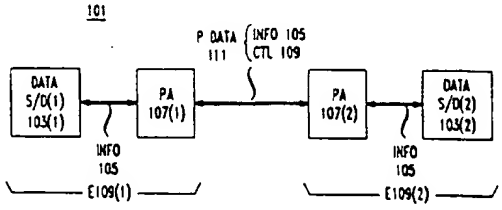
(72) Inventor : **Holzmann, Gerard Johan**
**15 Chestnut Hill Drive**
**Murray Hill, New Jersey 07974 (US)**

(74) Representative : **Watts, Christopher Malcolm Kelway, Dr. et al**
**AT&T (UK) Ltd.**
**5, Mornington Road**
**Woodford Green Essex, IG8 0TU (GB)**

## (54) Apparatus and methods for implementing protocols.

(57) Apparatus and methods for communicating using protocols. The apparatus and methods employ protocol descriptions written in a device-independent protocol description language. A protocol is executed by employing a protocol description language interpreter to interpret the protocol description. Communication using any protocol for which there is a protocol description may be done by means of a general protocol. The general protocol includes a first general protocol message which includes a protocol description for a specific protocol. The protocol apparatus which receives the first protocol message employs a protocol description language interpreter to interpret the included protocol description and thereby to execute the specific protocol. The protocol apparatus may also be made to adapt to its environment by encaching protocol descriptions which were received in an earlier first general protocol message and interpreting an encached protocol description in response to a second general protocol message which includes a protocol identifier specifying the encached protocol description.

FIG. 1




EP 0 555 997 A3